Managing Program Binaries in a Heterogeneous Unix Network

Managing Program Binaries In a Heterogeneous UNIX Network

Paul Anderson – University of Edinburgh

ABSTRACT

This paper presents some of the techniques adopted in the Computer Science Department at the University of Edinburgh for providing a consistent user environment across a large network of heterogeneous workstations. These include system management techniques that allow non-privileged users to maintain and install network-wide application packages, as well as software and techniques for automatically distributing and replicating program binaries across the network.

Background

In a highly distributed network, it is often desirable to provide a consistent user environment across all workstations, so that users may move freely between systems, and the network can evolve with a minimum of disruption to the service. The extreme approach is illustrated by Project Athena at MIT [1] where an identical operating system, including the kernel and a large quantity of local software, is used on all workstations. Whilst this provides a totally uniform environment and good control over the available facilities, it is not suitable for many sites because of the difficulty of supporting this amount of software, especially where many different architectures are involved. This is a particular problem where there is a need to regularly adopt new and different hardware for technical, financial, or political reasons.

The system employed in the Computer Science Department at Edinburgh University, [2] involves a compromise whereby a minimal base operating system, supplied by the hardware vendor, is overlaid with a standard local environment providing the higher-level facilities such as the shell, the window system, the editor, and other applications. [1] Clearly, this does not provide an absolutely identical environment across all platforms, but it does allow new machines to be incorporated quickly and the integration subsequently improved gradually by porting more of the standard environment, as necessary. The manufacturer's system software and specific enhancements also remain available (although unsupported) for those who wish to use them.

The other essential component, in providing the user with a consistent view of the network, is a virtual, network-wide filesystem. Home directories, for example, are physically located on a server in the user's home cluster, but are always referenced as **/home/user** and can be accessed from anywhere on the network. The manufacturer's implementation of NFS [3] together with the AMD automounter [4] provides a basis for such a virtual filesystem that is portable across many different platforms. The AMD maps are currently provided via NIS [5] , but it is likely that these will be converted to Hesiod [6] in the future, allowing authority for a map to be delegated to the appropriate cluster. This use of standard NFS on a wide scale does incur several penalties, such as the need for a network-wide uid allocation scheme, and some difficult security issues which cannot be completely resolved without modifications to the NFS code itself. DNS [7] and NIS provide a global namespace for hosts and usernames.

Some filesystems, such as those containing home directories, are necessarily stored as a single live copy (since they need to be writable, and the traffic is relatively low). Other filesystems, however, such as the network-wide program binaries, need to be replicated across several servers, both to provide resilience against server failure and to distribute the load. The remainder of this paper presents some techniques that allow these *packages* to be maintained by users without superuser privileges, and system managers to control the distribution and amount of replication on a per-package basis. The basic aims are very similar to those of the *Depot* [8] framework, but a different filesystem organization, together with some local programs, provides a more flexible mechanism for controlling the distribution and replication of individual packages among servers.

Packages

Each *package* is allocated a user-id and all files belonging to that package are are created with the appropriate uid. The master distribution of the package is stored in the home directory, and any group of people can work on the package by changing their working uid to that of the package. A modified version of **su**, called **nsu**, allows users to change the working uid without supplying a password, providing that they are members of the the netgroup

[1] The GNU **bash** shell, MIT X11R4 and GNU **emacs**.

nsu_*package_name*. This allows systems managers to authorize users, simply by adding them to the appropriate netgroup in the NIS netgroup map. Direct logins with package uids are disabled in the password file, so that the "real" user is always identifiable. (The conventional approach of using Unix groups for this purpose was rejected for several reasons, including limits on the number of groups to which a user may belong, the need to make all package files group-writable, and differences in group semantics between different systems.)

Compiling packages for multiple architectures is often a problem because the object files created by a compilation for one architecture may interfere with those created on another architecture. Unless the package provides its own mechanism, this is usually handled by constructing *shadow trees* (a filesystem hierarchy identical to the master, but with each file replaced by a symbolic link to the master copy). The compilations are performed in the shadow directory, providing common source files, but separate object files for each architecture. The shadow trees are built with the utility program **lfu** (below) and are stored on a separate filesystem (**/obj/local/***architecture*) which, since it contains only transient files, does not require backup.

On a standalone system, the final object files would be installed under **/usr/local** in subdirectories such as **bin**, **lib**, etc., similar to the usual hierarchy under **/usr**. Files that are common to all (or several) architectures, such as manual pages, or fonts, are stored under **/usr/local/share**, possibly with a symbolic link from other directories, in a similar way to **/usr/share** under SunOS. No files are installed directly into other directories under **/usr**, because of the problems involved in reinstating these files when the system is upgraded, or a new system is installed (a practically continuous activity in a large network).

The use of separate uids for each package has several additional benefits:

- It is easy to locate all the files corresponding to a certain package by running **find** with the required package name on the **/usr/local** filesystem.
- The system manager can obtain summaries of the space occupied by each package, using **du**, or the local program **lfck** (see Appendix I) which checks the filesystem for files with suspect owners, as well as providing a detailed disk usage summary.
- A regular daemon collects **README** files from the home directories of all the packages, appending them together into a single document that provides a summary of all the packages on the system. Users can then browse this document and locate the source (or at least the master distribution) for any package simply by looking in the home directory.
- Mail directed to a package account can easily be forwarded to the user(s) responsible for the maintenance of the package.

Distribution and replication

The true network situation is more complex than the simple standalone model presented above, because servers need to supply binaries for more than one architecture, and multiple copies of the binaries need to be distributed among several servers. The general approach to this problem is to designate a *master* server for each package (usually in the home cluster of the user maintaining the package) which holds master copies of installed binaries for that package on all architectures. The *slave* servers then run a nightly job to update themselves from the various masters, and the clients mount **/usr/local** from a nearby slave carrying the appropriate architecture. Programs such as **rdist** [9] are designed to perform this type of update operation, but there are several problems which could not be solved adequately by existing software, and a local program **lfu** performs the server updates. Some of the important features include:

- The copying process should be as faithful as possible, including ownership and status of all types of filesystem object. For example, files with *holes* can be created by seeking past the end of the file; when these files are copied by most normal programs, the holes will be filled, usually generating a file larger than the source file.
- Given the large volume of software (currently over 1Gb for a single architecture and the common *shared* files), it is not generally possible for every slave to carry binaries for every package, so some mechanism is required to load easily configurable subsets of packages onto slave servers. However, to maintain a consistent view of the virtual filesystem, there must be some mechanism to ensure that files which are not resident on a particular slave are still accessible by the same pathnames.
- Slave servers will contain files from more than one master server, so it is essential that the set of files from one master server can be updated onto the slave without disturbing the set of files supplied from the other masters.
- Special actions are likely to be necessary when certain files are updated. For example, when replacing the binary for a daemon, it is essential that the existing binary is not immediately deleted, since it may be mapped into a running process. (It may however, be useful to automatically inform the system manager that the process needs restarting).
- A good log of all updated files is valuable, both for debugging, and to provide users with a list of files that have recently changed.

Since a network-wide filesystem is already supported, this can be used to access the master servers and no special network code is required in the update program.

One disadvantage of this nightly "bulk" updating of slave servers, is that the inconsistent state of the filesystem during the update could potentially cause problems for any programs running at the time. In practice, this has not proved to be a problem and any programs which are likely to be affected can be marked for special treatment (see the example for updating daemon programs below).

The virtual filesystem hierarchy

Ordinary users are normally only concerned with the **/home** and **/usr/local** directories from the virtual filesystem. **/home** provides the home directories and **/usr/local** provides access to binaries for all the local packages. Package maintainers install packages in **/export/local** which is the master server for the the current cluster.

The master and slave servers for any particular cluster are also accessible as **/export/remote/***cluster* and **/usr/remote/***cluster*. This allows the update program to retrieve the latest version of a package from the appropriate master server. Files belonging to packages that are not carried on a particular slave server can be replaced by symbolic links to a slave server in a cluster which does carry the package. In this way, common packages can be carried by all servers, but packages that are normally of interest only to one particular cluster, can be carried on the slave servers from that cluster only. (although they are still accessible from everywhere else, under exactly the same pathnames, because of the symbolic links).

Figure 1 illustrates two clusters, each containing a master, a slave and one client:

- Package A is maintained on the master server in cluster A, and is copied onto the slave servers for both clusters.
- Package C is a specialist package for the users in the **vlsi** cluster. It is copied onto the **vlsi** slave server, but links are inserted into the local slave server so that the package is usable from the client of the local slave. Note that the actual value of the links will be **/usr/remote/vlsi/sun4/....**; the name **/disk/local** refers only to the mount point of the disk containing the binaries on the slave server (it is not

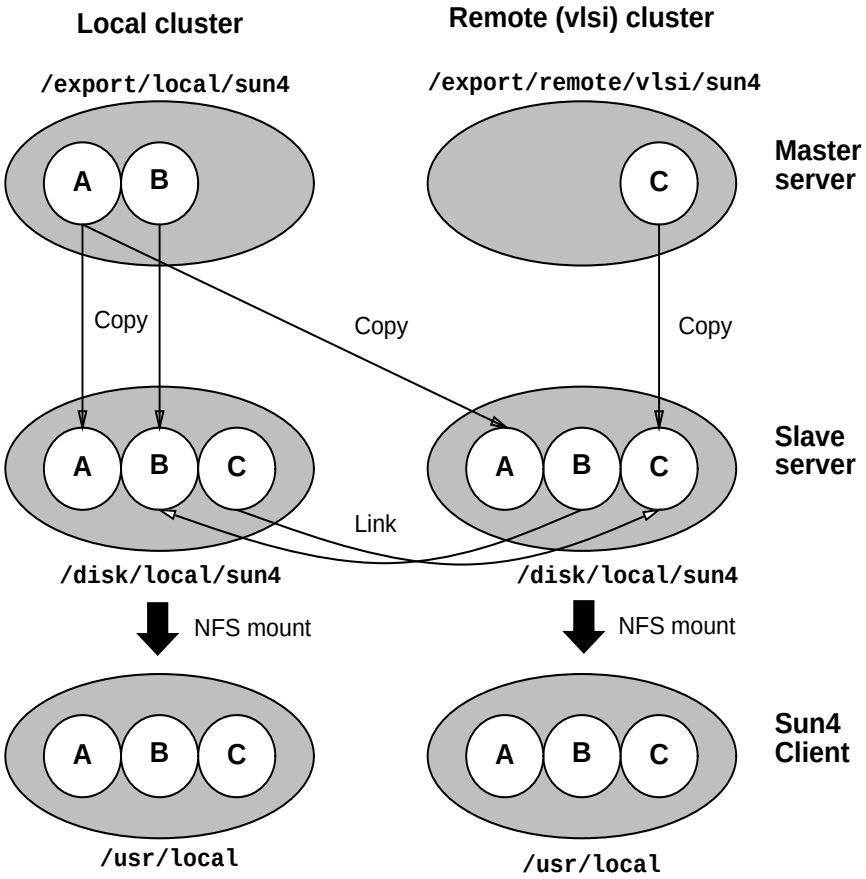

Figure 1: Updating servers

part of the network-wide filesystem).

- Package B is a similar specialist package for the local cluster. When this is referenced from remote slave, then the full name of the local cluster would need to be specified in the links.

Notice that all architectures are visible on the servers, but each client sees only its own architecture under **/usr/local**, and this view is functionally identical for every client.

Resilience

It is very important that the interdependence between machines is clearly defined and controlled; on a single machine, even links from public packages into users home directories, for example, could go unnoticed. In a distributed environment, this causes complex interdependencies and it is easy to reach a situation where a single workstation is dependent on too many servers and will fail when any one of the servers fails.

In most cases, the automounter is configured to mount **/usr/local** from the bootserver of the workstation, so that the workstation depends only on a single server for its basic operation. Where the workstation has no bootserver (or where the bootserver does not carry local binaries), the automounter will usually choose between several "nearby" (i.e., on the same wire) servers that carry the binaries for the appropriate architecture. Files that do not reside on the local slave are linked to remote slave servers, where there is also a choice between more than server. This arrangement has the following properties:

- A diskless workstation depends only on its bootserver for the basic operating system and most of the programs from the local environment.
- Other workstations can obtain these programs from more than one server.
- All of the basic local environment, and all applications which are of particular interest to this cluster, are provided directly by one of the above local servers. Other programs are provided from a remote slave server, although the remoteness is transparent to the user, and multiple copies are still available to provide resilience.
- Without the use of disk, or server, "mirroring", home directories always form a single point of failure (although on a per-user, rather than a per-workstation basis). However, in the event of a server failure, these can normally be brought online again very quickly by moving disks, or restoring onto a different machine.

The lfu program

This program is responsible for copying files from a master server onto a slave server, implementing all of the features mentioned above, such as replacing certain files with symbolic links to other servers. In its basic mode of operation, **lfu** traverses the hierarchy of the master server in parallel with the hierarchy of the slave server. Files on the slave are deleted and/or copied from the master to make the slave into a faithful copy of the master. This basic operation can be modified by providing a *script* to **lfu** containing a list of *conditions* and *actions*. The *actions* are applied to any file matching the *conditions*, in place of the default action. For example:

```
owner=vlsitools {
  link;
  source=/usr/remote/vlsi;
}
```

will update all files on the slave server from the master server, except that files (or directories) owned by the package **vlsitools** will be replaced with links to the corresponding file on one of the servers in the **vlsi** cluster (automounted under **/usr/remote/vlsi**).

The use of ownership to identify the packages avoids the need to specify large explicit filesets. It also allows the netgroup mechanism to be used for specification of package sets. For example, if **server-A** updates with

```
owner=@bfiles {
  link;
  source=/usr/remote/server-B;
}
```

and **server-B** updates with

```
owner!=@bfiles {
  link;
  source=/usr/remote/server-A;
}
```

then, the servers A and B will hold disjoint sets of packages with all the packages in the netgroup **bfiles** being held on the **server-B** and all other files being held on **server-A**. Simply adding a new package to the netgroup will cause all files belonging to that package to migrate from one server to another when the updates run. In practice, multiple servers will be involved and the package would migrate off all the servers in one cluster onto the servers in another cluster.

This provides a powerful mechanism for the system manager to control the usage of disk space on the slave servers and the possibility of extending the facility to provide a cacheing behaviour is currently being investigated — the following script would migrate files that had been accessed in the last two weeks onto the local server and replace less

frequently used files by a link to the copy on a larger server:

```
access > 2wks {
  link;
  source=/usr/remote/mainserver;
}
```

The following examples show some other features of the **lfu** program:

updating daemon programs

If the file **daemons** contains the names of the daemon programs, then the script:

```
path=@daemons {
  exec restart $F;
  keep;
}
```

will execute the shell script **restart** (with the name of the updated program as a parameter) whenever one of these programs is updated[2]. This shell script could automatically restart the daemon or mail the system manager to intervene manually. The **keep** action indicates that the old version of the program on the slave server should be renamed, rather than being deleted, since the text may be mapped into a running program. (In this example the renamed version of the file would be deleted automatically next time the update runs, since the corresponding master file would not exist).

Programs that must be owned by root

Some programs need to be owned by root, perhaps because they need to run "setuid". If such files exist on the master server, they can be difficult to identify and can cause problems across NFS if the **root** user is mapped onto **nobody**. The following script will change the ownership and permissions of any files specified in **rootfiles**, as they are copied. Any files owned by root on the master will generate an error message and not be copied; this ensures that the **rootfiles** are the only root-owned files on the slave server, providing a useful security check.

```
path=@rootfiles {
  chown root; chmod u+s;
}
else owner=root {
  error "Root file on master";
}
```

[2]Unfortunately, the stateless nature of NFS normally makes it impossible to automatically detect files which are currently in use.

Updating from multiple servers

When a slave server updates from more than one master server, the files supplied by one of the masters must not be deleted when updating from the other. For example, the following script could be used to update the local slave server of Figure 1 from the local master:

```
owner=package-C {
  ignore;
}
```

When updating from the remote **vlsi** master, the following script could be used:

```
owner=package-A |
owner=package-B {
  ignore;
}
else owner=package-C {
  link;
  source=/usr/remote/vlsi/sun4;
}
```

Netgroups are particularly useful here to define the sets of packages supplied by each of the master servers, and the sets of packages carried by each slave.

Some difficulties

The following paragraphs illustrate some issues that have required special attention:

Package Installation

Care is needed when installing packages, since the binaries must be installed on the master server (**/export/local/**), but any references made by the running programs must be made to files on the current slave server (**/usr/local**). Most installation procedures are not designed to handle this situation, and it is easy to inadvertently install packages that make direct references to the master server when they are running. Such programs will continue to operate, but this introduces an unwanted dependency between the client and the master server. It may also introduce an excessive load on the master server. These dependencies are not always easy to detect; sometimes the automounter can be seen unexpectedly mounting a directory from the master, or, more usually, the dependency is only noticed when the master server is shut down.

Another difficulty for the package maintainer is the time delay between installing a package onto the master and having it propagate to the slave, where it can be tested. For complex installations, a particular workstation may be configured to reference certain binaries directly off the master server, so that the installation can be tested without waiting for changes to propagate to the slaves[3]. The state of a particular

[3]Forcing updates is also possible, but this leaves slave

package can be "frozen" on all the slave servers by added the package name to a netgroup which is ignored by the **lfu** scripts. This is useful when working on complicated installations or version updating, to prevent incomplete or inconsistent installations being propagated.

Conflicting filenames

With a large number of packages available, **/usr/local/bin**, for example, becomes very large and there is an increasing chance of the same filename being used by more than one package. Where a package (such as **X11**) has a large number of associated binaries, these are often moved to a subdirectory and arrangements made to include the subdirectory in the user's path, when appropriate. Normally, however, if two packages from different master servers include a file with the same name, the conflict may not be immediately apparent, although it should be possible to detect.

Updating multiple slaves

With a large number of slave servers, there can be a problem with too many slaves attempting access the same master at the same time. Currently, this is prevented by having some slave servers update from other slaves, rather than directly from the master. This idea could be extended to a hierarchy of slaves, which would prevent any one particular server becoming overloaded.

In a typical update run, 80-90% of the cpu time (and NFS traffic) generated by **lfu** is incurred in scanning the filesystems to locate files which have been changed. In the case of identical slave servers, it should be possible for just one of the servers to perform the scan, and pass on information about the required updates to the others. This is currently being investigated.

Conclusions

The techniques described above have evolved over the past three years on the network within the Computer Science Department, and have recently been extended to include clusters belonging to other small groups. Currently, 200-300 workstations are supported with three master servers and some tens of slave servers. Four major architectures are supported[4], and several others are included with a lesser degree of support.

In practice, the network is continually evolving and there are always some clusters and individual machines that are only partially incorporated. Certain clusters may decide (perhaps for licencing reasons) not to carry a particular package at all, or not to provide access to a particular group of home directories (perhaps for security reasons). The ability to support this degree of flexibility at the same time as providing a consistent and stable user environment has been one of the most important benefits.

The concept of providing a uniform environment across a heterogeneous network has undoubtedly been successful, and is popular with users. The need to attempt this without modifications to the hardware vendor's base operating system has lead to some obvious visible differences between different platforms and many difficulties that could have been avoided by running a completely standard system. However, a reasonable compromise has been reached and new hardware can usually be incorporated, with an acceptable degree of integration, very quickly.

The method adopted for management of software packages has generally been very successful, on the present scale. System managers are usually unaware of the detailed changes to individual packages, but are able to monitor and control the placement of the binaries very easily, whilst users are unaware of the underlying services and can use any software from any workstation.

The success of the current system is leading to its adoption by other clusters and, although we expect the basic concepts to scale reasonably well, the wider scale is expected to emphasize the difficulties of using standard available software, such as the vendor's implementations of NFS. As these kind of problems become more widespread, we hope that vendors will begin to incorporate solutions (such as the kerberos [10] enhancements to NFS) into their own products.

Acknowledgements

The implementation and evolution of the network would not have been possible without the continual efforts of all the systems staff in the Computer Science Department. In particular, Alastair Scobie and Russ Green, have been actively involved in the design of many of the concepts discussed above.

Biography

Paul Anderson graduated in Pure Mathematics from the University of Wales in 1977. He taught Mathematics and Computer Science at the North East Wales Institute of Higher Education until 1984 when he became system manager for the Institute, establishing a new computer centre and software development team. In 1988 he moved to the University of Edinburgh as Systems Development Manager with the Laboratory for the Foundations of Computer Science, where he is currently managing the laboratory network and working with other system managers to improve the integration and

servers in inconsistent states and can be rather slow.

[4]Sun SPARC (SunOS), Sun 68000 (SunOS), HP9000 (HP/UX) and DECstation 5000 (Ultrix).

administration of the university networks. Paul can be reached by mail at the Laboratory for the Foundations of Computer Science; Department of Computer Science; University of Edinburgh; King's Buildings; Edinburgh; EH8 3JZ; U.K. Reach him electronically at paul@dcs.ed.ac.uk.

References

1. Jennifer G. Steiner and Daniel E. Geer, Network services in the Athena environment, Project Athena, Massachusetts Institute of Technology, Cambridge, MA 02139.
2. Paul Anderson, Installing software on the Computer Science Department network, Department of Computer Science, University of Edinburgh, Edinburgh, August 1991.
3. Sun Microsystems, ''Network File System: Version 2 protocol specification,'' in *Network programming guide*, pp. 168-186, Sun Microsystems, 1990.
4. Jan-Simon Pendry, AMD - An automounter, Department of Computing, Imperial College, London, May 1990.
5. Sun Microsystems, ''The Network Information Service,'' in *System and network administration*, pp. 469-511, Sun Microsystems, 1990.
6. Stephen P. Dyer, The Hesiod name server, Project Athena, Massachusetts Institute of Technology, Cambridge, MA 02139.
7. Sun Microsystems, ''Administering the Domain name service,'' in *System and network administration*, pp. 513-554, Sun Microsystems, 1990.
8. Kenneth Mannheimer, Barry A. Warsaw, Stephen N. Clark, and Walter Rowe, ''The Depot: A framework for sharing software installation across organizational and UNIX platform boundaries,'' *Proceedings of LISA IV Conference*, 1990.
9. Sun Microsystems, ''rdist (1),'' in *SunOS reference manual*, Sun Microsystems, 1990.
10. Jennifer G. Steiner, Clifford Newman, and Jeffrey Schiller, Kerberos: An Authentication service for open network systems, Project Athena, Massachusetts Institute of Technology, Cambridge, MA 02139.

Appendix I: lfck

The following example shows a fragment of output from the **lfck** program summarizing the disk usage (by package, and by architecture) on one of the master servers. The figures in brackets represent the space occupied by files exported to the slave server, and the main figures represent space occupied on the master server only (home directory of the package, and compilations under **/obj/local**). Notice that this is only a section of the real output, so the row and column totals do not correspond to the figures in the table.

Package	share	sun3	sun4	Total Mb
X11 Release 4	24.1 (24.7)	36.8 (29.9)	36.0 (30.2)	615.6
Poplog	139.7 (29.9)	25.2 (28.9)	0.1 (27.1)	297.1
GNU Emacs	185.9 (11.1)	0.1 (11.6)	0.1 (11.0)	251.5
TeX	85.3 (41.5)	22.5 (17.6)	24.4 (13.3)	233.1
InterViews	35.7 (1.7)	9.2 (6.4)	28.8 (18.7)	113.8
GNU C Compiler	30.7 (2.1)	10.3 (2.0)	11.4 (2.0)	110.5
Centaur	52.4	11.4 (14.5)	0.1 (18.7)	96.8
Modula 3	41.2 (0.1)	0.0	31.8 (12.7)	94.6
Generic graphics	7.7 (0.8)	17.0 (15.0)	11.3 (4.2)	89.1
IE Editor	0.1	0.0	0.0	0.1
Local Admin Data	0.1	0.0	0.0	0.1
Total Mb	2621.8 (235.2)	288.5 (318.7)	282.2 (370.5)	4742.7

lfck can also detect files which are not owned by a valid package and apply a number of heuristics to suggest the correct owner.

Appendix II: lfu

lfu currently accepts the following *conditions* and *actions*:

Conditions:

name=*value*	Matches the name of the file against a regular expression. An explicit list of files can also be specified.
path=*value*	Matches the pathname of the file against a regular expression. An explicit list of pathnames can also be specified.
owner=*value*	Tests the owner of the file. A netgroup can also be specified.
group=*value*	Tests the group of the file.
type=*value*	Tests the type (mode) of the file.
age*[><=]value*	Tests the age (mtime) of the file.
access*[><=]value*	Tests the last access time of the file.

Actions:

update	The default action - update the file if the source is more recent than the destination.
ignore	Ignore the file.
delete	Delete the file.
preserve	Update, if necessary, but do not delete.
chmod *mode*	Change the mode (perms) of the file.
chown *owner*	Change the owner of the file.
chgrp *group*	Change the group of the file.
link	Link objects rather than copying.
shadow	Copy directories, but link other objects.
source *value*	Specify the source directory for links.
log	Log any changes to this file.
logall	Log any examination of this file.
error *msg*	Report specified error message.
exec *command*	Execute specified command whenever file is updated.
keep	Rename file rather then deleting.
fill	Do not attempt to duplicate "holes" in files.
newtime	Do not duplicate the mtime when copying files.
force	Update files regardless of the file times.
follow	Follow symbolic links.